Erschienen in:
Brand-Pook, H.; Fleer, A.; Spitta, T.; Wattenberg, M. (Hrsg.): Nachhaltiges Software Management. Lecture Notes in Informatics P-209, Bonn, 2012, S. 135-152

# Entwicklung und Betrieb eines Campus-Management-Systems

## – Aspekte zur Nachhaltigkeit am Beispiel TISS –

Thomas Grechenig[1)], Thorsten Spitta[3)], Monika Suppersberger[1)], Wolfgang Kleinert[2)], Ronald Steininger[1)], Christof Kier[1)], Martina Pöll[1)]

[1)] TU Wien, Industrielle Softwaretechnik,
{thomas.grechenig, monika.suppersberger, ronald.steininger, christof.kier, martina.poell}
@inso.tuwien.ac.at

[2)] TU Wien, Zentraler Informatikdienst,
wolfgang.kleinert@zid.tuwien.ac.at

[3)] Universität Bielefeld, Angewandte Informatik/Wirtschaftsinformatik
thSpitta@wiwi.uni-bielefeld.de

**Abstract:** Hochschulen stehen wie alle großen Institution stärker als früher vor der Herausforderung, Prozesse der Lehre, Forschung und Administration mit geeigneten IT-Mitteln effizienter machen zu müssen. Der folgende Beitrag erläutert anhand des Fallbeispiels von TISS, dem Campus-Management-System (CaMS) der TU Wien, Aspekte zu nachhaltiger Einführung und Betrieb eines solchen Systems. Zusätzlich zeigt der Beitrag Kernfaktoren auf, die für die Ablöse von Altsystemen und die Einführung eines modernen und zukunftssicheren CaMS elementar sind.

## 1 Problemstellung und Ziel

*„Der Leidensdruck deutschsprachiger Hochschulen im Bereich Campus-Management (CM) nimmt derzeit permanent zu“* [Bo09, S. 451]. Seit dem Desaster um das Zulassungssystem der deutschen Hochschulen ist auch der Öffentlichkeit deutlich, dass die Einschätzung „nimmt zu“ von 2009 richtig war. Der Anlass für das „Leiden“ ist schnell benannt. Seit dem Übergang auf BA/MA-Studiengänge mit studienbegleitendem, iterativem Prüfen kann keine Hochschule mehr dies mit dem althergebrachten Instrument Excel-Tabellen bewerkstelligen. Die Universitäten brauchen heute für ihre Kernprozesse buchende Systeme, die die Daten der vielen Vorgänge transaktions- und rechtssicher (> 10 Jahre nicht änderbares Archiv) verwalten. Für diesen Komplex spricht man gesamtheitlich von *Campus-Management-Systemen* (CaMS), unter denen es gegenwärtig noch keines gibt, das den Status *Standardsoftware* beanspruchen könnte [BGS10].

Auf der Tagung WI2009 in Wien [Bo09] gab es Berichte der Universitäten Göttingen, Hamburg, Osnabrück, Bielefeld [Br09] und Darmstadt, die zeigten, dass viele Akteure sich bemühen, wenigstens mit behelfsmäßigen Lösungen zu „überleben“. Auf der Tagung *Software Management* im Dezember 2010 in Aachen wurde erstmals im deutschsprachigen Raum ein normatives Konzept vorgestellt, was ein CaMS überhaupt sein könnte und eine kurze Bestandsaufnahme zu Projekten geliefert, die auf ein Standardsystem abzielen [BGS10].

In diesem Beitrag wird das in [Gr10, S. 91ff.] als sehr anschauliches Beispiel, damals "HISS" genannte System, gezeigt und seine Entwicklung beschrieben. Es befindet sich inzwischen als TISS (*TU Wien Informations-Systeme und Services*) vollständig im Echtbetrieb für 32.000 *Core User* (Beschäftigte + aktiv Studierende) und geschätzt mehreren 100.000 weiteren *Usern* (Interessenten, Absolventen und Öffentlichkeit) pro Jahr. Es wird als Beispiel für ein „nachhaltig“ betreibbares System vorgestellt. Darunter verstehen wir zunächst eher intuitiv *Langlebigkeit* bei gleichzeitiger *Ressourcenschonung*. Die Anforderungen an in diesem Sinne nachhaltige Softwaresysteme wollen wir unter vier Aspekten betrachten:

(1) Methodische Migration der Daten aus den Altsystemen und Archiven

(2) Software-Infrastruktur (Datenbasis, Framework, Komponenten-Konzept)

(3) Einbeziehung der Stakeholder in Entwicklung und Evolution des Systems

(4) Betriebskonzept.

Besonders der letzte Punkt muss bei der Entwicklung und Evolution von Software heute umfassend bedacht werden, obwohl er in der „schulmäßigen“ Lehre zum Software Engineering nach wie vor kaum vorkommt. Langlebige Softwaresysteme bestehen heute beobachtbar aus einem entwicklungszentrierten Teil und einer IT-betrieblich zentrierten Serviceumgebung, die aus dem Gesamtintegrations- und Betriebsbedarf besteht und letztlich nie frei von Funktions- und Applikationsaspekten ist. Egal wie „sauber“ die Architektur des Systems zu Beginn etabliert wird, die Volatilität der informationstechnischen Evolution „erzwingt“ diese Drift hin zur Betriebstechnik.

## 2 Ausgangssituation

Seit 1968 setzt die Technische Universität Wien IT-Systeme zur Unterstützung der administrativen Tätigkeiten ein. Das erste Verwaltungssystem namens TUWIS (TU Wien Informationssystem) war eine COBOL-Applikation, die vorrangig die Administration von Studierendendaten, deren Studien und das Erfassen von Zeugnissen ermöglichte. Die Weiterentwicklung des Systems war über Jahrzehnte von der Implementierung neuer Funktionen „auf Zuruf“ und Fehlerkorrekturen geprägt. Trotz einer grundlegenden Umstellung der Datenhaltung von Dateien auf Oracle-Tabellen und den dafür nötigen Softwareanpassungen sah man weder auf Daten- noch auf Applikationsebene den Bedarf für ein umfassendes, integriertes Redesign des Systems. Nachdem TUWIS über Jahrzehnte

hinweg ausschließlich von Fachabteilungen der Universität genutzt wurde, konnte es Ende der 90er Jahre um eine Webapplikation ergänzt werden, so dass auch Studierenden erste Funktionen, beispielsweise die Möglichkeit zur Prüfungsanmeldung, zur Verfügung standen. Mit der Neuentwicklung dieser Webapplikation unter der Bezeichnung TUWIS++ 2003 wurden erste Stimmen laut, die mittlerweile stark gewachsene und mit anderen Systemen der TU Wien verstrickte COBOL-Basis zu ersetzen. Die Realisierung wurde jedoch nie in Angriff genommen; die Systeme wuchsen weiter. Das Ergebnis war ein wartungsintensives Flickwerk, bestehend aus inhomogenen und stark interdependenten Teilsystemen mit komplexen und stark verwachsenen Strukturen. Datenhaltung, Anwendungslogik und Präsentationsebene waren nur sehr mangelhaft getrennt, Dokumentation war kaum vorhanden oder unzuverlässig. Eine Etablierung von verlässlichen Software Engineering Methoden war unmöglich.

Die TU Wien befand sich damit in der Situation, dass auch die Anpassung interner, organisatorischer Prozesse auf Grund der fehlenden Beweglichkeit der IT-Systeme kaum noch möglich war. Um diesen Mangel zu beseitigen, entschied man sich Ende 2007 für eine Neuentwicklung der Systeme in Eigenregie mit Unterstützung eines erfahrenen Entwicklungshauses unter dem Projekttitel TISS (Kurzbezeichnung für „TU Wien Informations-Systeme und Services“). Ziel des Projekts war *„die Etablierung einer langlebigen IT-Strategie, die*

- *eine gemeinschaftliche technische Architektur bereitstellt,*
- *ein modernes interagierendes Applikationsmanagement erlaubt,*
- *die Altsysteme schrittweise ablöst, wo, wenn und wann dies zweckmäßig ist,*
- *die langfristige Wartung sicherstellt,*
- *das Einpflegen neuer Dienste organisch bereitstellt,*
- *benachbarten Systemen einen qualifizierten Docking-Partner anbietet und diesen als kompaktes Leitsystem dient.“* [Kl08]

Das wörtliche Zitat aus der Universitätszeitung zeigt, dass die Einbeziehung der Stakeholder von Anfang an aktiv betrieben wurde.

## 3 Das Migrationsproblem: Altdaten und -systeme

Bei jeder Systementwicklung setzt man schon lange auf den Daten von Altsystemen auf. Dieses Problem der Migration von Daten wurde eindringlich von dem Schweizer IBM-Berater Max Vetter, habilitiert 1982 an der ETH Zürich, in einer Reihe weit verbreiteter Bücher problematisiert (z.B. [Ve90]). Vetters Veröffentlichungen basieren auf den Erfahrungen mit den Datenbeständen vieler Kunden der IBM, darunter mehrerer Schweizer Großbanken (s. auch, recht aktuell aus der Credit Suisse, [MWF08]).

Alle seine Bücher leitet Vetter, seit der ersten Auflage 1982, mit dem folgenden Satz ein:

*„Das Jahrhundertproblem der Informatik besteht in:*

1. *Der Bewältigung des Datenchaos, das infolge unkontrolliert gewachsener Datenbestände fast überall entstanden ist. [2. …]“* [Ve90, S. 5]

Vetter mag noch 1990 geglaubt haben, das Problem sei im 20. Jahrhundert lösbar. Heute wissen wir, dass dies noch immer nicht zutrifft.

Bei der Konzeption von TISS wurde das Problem erkannt und bearbeitet. Einer der ersten Schritte zu Beginn des Projekts war die Analyse und (Re-)Dokumentation der Datenstrukturen und -inhalte, um darauf basierend ein konzeptionelles Datenmodell für das neue Zielsystem TISS entwickeln zu können. Insgesamt waren in den Altsystemen TUWIS und TUWIS++ beinahe 350 Tabellen mit über 6500 Spaltendefinitionen vorhanden, die detailliert analysiert werden mussten. Die einzig valide Ressource für eine derartige Analyse war die Produktivdatenbank. Den Analyseprozess maßgeblich erschwert und zugleich umso mehr erforderlich gemacht haben insbesondere die folgenden vier Punkte:

(1) Veraltete oder fehlende Dokumentation: Die Dokumentation der Datentabellen und -attribute war nur sehr spärlich vorhanden und derart veraltet, dass keine zuverlässigen Schlüsse daraus gezogen werden konnten. Auch die Namen der Tabellen und Attribute waren wenig hilfreich bei der Interpretation möglicher Inhalte, da eine systembedingte Beschränkung auf 8 Zeichen die Namensgebung limitiert hatte (Bsp.: Studenten-Stammdaten wurden in zwei Tabellen mit den Namen "STFIX" und "STVAR" abgelegt).

(2) Fehlende Normalisierung: Bei der Umstellung von dateibasierter Datenhaltung auf eine relationale Datenbank in den 80er Jahren erfolgte kein Redesign der Datenstrukturen. Das Datenmodell entsprach daher in keiner Weise den gängigen Standards der Datenmodellierung, Normalisierungen wurden nie durchgeführt. Neben der vielfach redundanten Datenhaltung gab es auch keinerlei Fremdschlüssel-Definitionen (referenzielle Integritäten). Relationen zwischen Tabellen wurden nur in der Logik des Altsystems aufgelöst.

(3) Intransparentes Datenmodell: Zu Beginn des Analyseprozesses war nicht ersichtlich, welche Tabellen noch benutzt wurden und welche nicht. Zusammen mit fehlender Dokumentation und den vorherrschenden Namenskonventionen war dies ein großer Unsicherheitsfaktor.

(4) Unklare Datenhoheit[1]: Viele in TUWIS und TUWIS++ verwendeten Daten wurden zusätzlich in weiteren Systemen ver- und bearbeitet, häufig in separaten Datenbanken. Meist war unklar, welches System das führende und damit ver-

[1] Mit *Datenhoheit* fassen wir die strenge Koppelung der Operationen von Modulen mit den verwalteten Datentypen im Sinne der Objektorientierung und die klare Zuweisung der Änderungsrechte an Organisationseinheiten (Datenverantwortung) zusammen.

antwortlich für die Datenkonsistenz war. Die organisatorischen Datenverantwortlichkeiten waren diffus (s. [SB08, Kap. 8]).

Um unter diesen Voraussetzungen die nötigen Informationen über die Datenstrukturen und deren Inhalte zu erhalten, wurde ein eigens konzipiertes Verfahren angewandt. Im ersten Schritt wurden alle Bestandsdaten auf der Granularität von Tabellen analysiert und die Tabellen nach zwei Gesichtspunkten kategorisiert. Einerseits erfolgte eine fachliche Zuordnung zu definierten Themenkreisen (Personal- und Studentenstammdaten, Studium, Lehrveranstaltungen, Prüfungen, ...), andererseits wurde eine Unterscheidung nach der Relevanz der Daten getroffen. Tabellen, die nicht mehr in Verwendung waren oder keine geschäftsrelevanten Daten beinhalteten (z. B. temporäre Tabellen für Batch-Prozesse) konnten ausgeschieden und dadurch deren Umfang für eine weitere und tiefer greifende Analyse drastisch reduziert werden. Automatisch generierte Basisdokumentation über vorhandene Strukturdefinitionen und Wertebereiche unterstützten die anschließende Dokumentation der Geschäftsdaten und ermöglichten ein frühes Erkennen potentieller Konflikte mit geplanten Constraint-Definitionen. Zur besseren Abschätzung des Transaktions- und Datenvolumens wurde zusätzlich auch in einem 6 Monate andauernden Analyse-Zeitraum die Interaktion des Altsystems mit den Tabellen auf Basis von Datenbank-Statistiken analysiert. Aus den so gewonnenen Rohdaten konnte die Entwicklung des Datenbestands als auch die Lastverteilung (nach Tagen oder Wochen) aufbereitet werden. Details zur gewählten Vorgehensweise und den Ergebnissen sind in [St09] dargestellt.

Auf Basis der gewonnenen Information mussten also in einem mühsamen Prozess alle Attribute aller alten zu migrierenden Datenobjekte gegen das Soll-Datenmodell abgeglichen werden. Dieser Match verlangt eine exakte Spezifikation für jedes Attribut, und zwar für ein Programm, das zwar viele Male im Test läuft, aber nur genau einmal vor der Inbetriebnahme der neuen Datenbasis.

Entwicklungen, die diesen Aspekt vernachlässigen, können in keinem Fall *nachhaltig* genannt werden, selbst wenn die neue Software noch so elegant oder jahrelang positiv bewährt ist (Beispiel Standardsoftware). Warum? Systeme, die auf korrupten Daten aufsetzen, sind nicht nur unbrauchbar, sie sind ganz einfach falsch, weil sie „Informationen“ aus falschen Daten erzeugen.

Die Problematik wird am besten anhand eines Beispiels illustriert: Die Matrikelnummer eines Studierenden gilt zwar als eindeutig und nicht veränderbar, durch Fehler bei der Immatrikulation kam es aber regelmäßig zu Änderungsbedarf der TUWIS und TUWIS++ Datenbestände. Eine Änderung der Matrikelnummer zu einem Zeitpunkt, zu dem bereits Prüfungsanmeldungen, Zeugnisse und sonstige Daten im System gespeichert waren, erforderte eine manuelle und fehleranfällige Datenkorrektur an vielen verschiedenen Stellen. Inkonsistenzen waren vorprogrammiert. Verschärft wurde die Problematik durch unzureichende Validierungsmechanismen, wodurch beispielsweise die kürzeste erfasste Studiendauer eines Studierenden einen negativen Wert, die längste Studiendauer mehrere hundert Jahre betrug. So anekdotisch verhaltensoriginell dieses Beispiel auch sein mag, es ist typisch für viele in den 80er Jahren entstandene IT-Systematiken, die nie ganz abgeschaltet oder einem vollständigen Reengineering unterzogen wurden.

Für dieses gewichtige Problem der Informationsgenerierung auf Basis korrupter Daten aus eigentlich abgelösten Altsystemen ist bezeichnend, dass außer in [Gr10, Abschn. 8.3] die Datenmigration in heutigen Lehrbüchern der Informatik und der Wirtschaftsinformatik nicht ([So01], [Pf01], [BD04], [AGW05]) erwähnt wird.

Dank des intensiven Analyseprozesses zu Beginn konnte dieses Problem bei der Entwicklung und Einführung von TISS zufrieden stellend gelöst werden. Inkonsistenzen wurden lange vor der Inbetriebnahme des Systems aufgedeckt und geeignete Algorithmen entwickelt, um fehlende oder fehlerhafte Attribute im Rahmen der Migration zu ergänzen bzw. zu korrigieren.

# 4 Kurzbeschreibung des Systems

## 4.1 Überblick

TISS wurde als umfassendes Campus-Management-System für die TU Wien konzipiert. Die Autoren bedienen sich hierbei der Definition eines CaMS gemäß [Br09] als ein Softwaresystem, das die relevanten Prozesse einer Hochschule operativ unterstützt und der Führung daraus geeignete Informationen liefert. Die Grundfunktionen von TISS werden in Abschnitt 4.2 zusammenfassend dargestellt.

Abbildung 1 zeigt die Einbettung des CaMS in die Systemlandschaft der TU Wien. Der Umfang der CaMS Funktionalitäten lässt die Dimension des Projekts erkennen. Dem Bedarf nach wirksamer Unterstützung aller Prozesse aus den Bereichen Lehre, Organisation und Forschung ist umfassend Rechnung getragen.

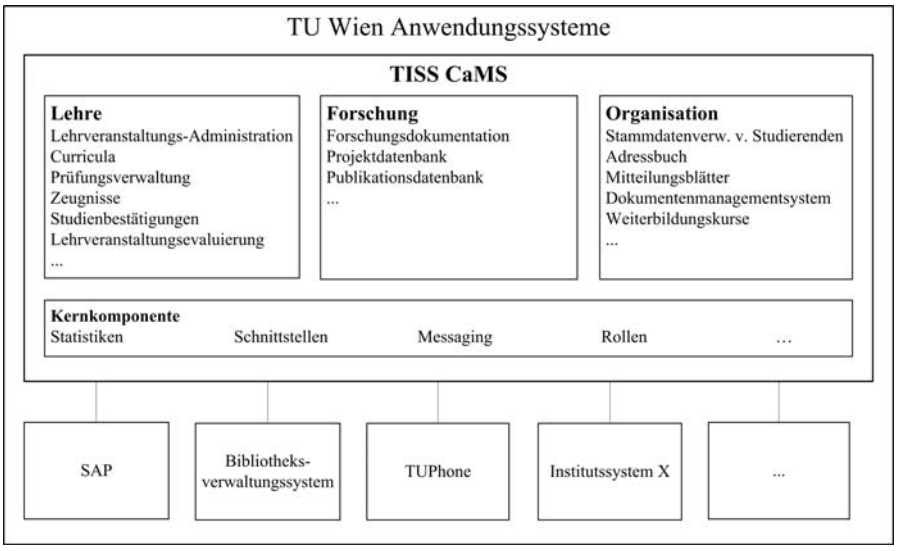


Abbildung 1: TU Wien Anwendungssysteme

Die saubere Definition und Integration von erweiterbaren und standardisierten Schnittstellen zu Drittsystemen, wie zum Beispiel zu SAP, zum Bibliotheksverwaltungssystem oder zur Steuerung der VoIP-Telefonie („TUPhone“), war ein zentrales Ziel von TISS. Anbindungen zu verschiedenen, oftmals auf Standardsoftware basierenden Systemen müssen ebenso leistungsfähig und änderungsstabil gebaut sein, wie das Kernsystem selbst.

### 4.2 Fachliche Struktur

Die fachlichen Prozesse der TU Wien, die von TISS unterstützt werden müssen, lassen sich in die drei Bereiche *Lehre*, *Forschung* und *Organisation* gliedern.

Der Bereich *Lehre* umfasst vorrangig die Prozesse rund um die Verwaltung und Abwicklung von Lehrveranstaltungen auf Basis von Curricula und bietet dabei Unterstützung in der Prüfungsverwaltung, bildet Zulassungsbedingungen und Anmeldebeschränkungen ab, ermöglicht das Ausstellen und den Druck von Zeugnissen und Studien-Bestätigungen, erlaubt die Evaluation von Lehrveranstaltungen durch Studierende, das Reservieren von Hörsälen und vieles mehr.

Für die *Forschung* sollen den Mitarbeitern und Mitarbeiterinnen geeignete Hilfsmittel die geforderte Forschungsdokumentation erleichtern. Derzeit wird dies bereits durch eine Leistungs- und Projektdatenbank unterstützt. Die Entwicklung einer integrierten Datenbank zur Verwaltung von Publikationen wird 2013 folgen.

Der Bereich *Organisation* bietet neben der Stammdatenverwaltung und Zulassung von Studierenden vorrangig Funktionalitäten für die administrativen Services der TU Wien. Ein Adressbuch stellt Kontaktdaten der Mitarbeiter und Organisationseinheiten bereit, Mitteilungsblätter ermöglichen die Veröffentlichung relevanter Informationen der Universitätsleitung, ein Dokumentenmanagementsystem unterstützt die digitale Verwaltung von Akten und ein Weiterbildungskatalog bietet Mitarbeitern eine Auswahl zahlreicher Möglichkeiten zur Fortbildung.

Eine zusätzliche Komponente, hier als *Kernkomponente* bezeichnet, stellt zentrale und übergreifende Services wie Message-Funktionen und Statistiken bereit, definiert Benutzerrollen für Zugriffsberechtigungen und einiges mehr.

### 4.3 Softwarearchitektur

Die fachliche Unterteilung in einzelne Themenkreise findet sich in der Modularisierung der Softwarearchitektur wieder: TISS besteht aus vielen Komponenten, die jeweils für sich ein bestimmtes fachliches Thema bedienen. Zwischen ihnen ist die Kommunikation nur über genau definierte Schnittstellen möglich. Der Ansatz hat Vorteile sowohl in der Entwicklungsphase des Gesamtsystems als auch in der Wartung und Weiterentwicklung:

- Die Modularisierung auf technischer Ebene erzwingt, dass fachliche Bereiche strikt getrennt bleiben.

- Die Datenhoheit wird ebenso durch die fachliche Trennung sichergestellt. Die einzelnen Module sind auf diese Weise nur für einen überschaubaren Teil der im Gesamtsystem gespeicherten Daten zuständig.

- Die Module können unabhängig voneinander (weiter-)entwickelt werden, was gerade bei großen Systemen eine schrittweise Einführung erlaubt und so die planmäßige Ablösung von Altsystemen erst ermöglicht.

- Alle Komponenten sind einzeln lauffähig und kompilierbar: Entwickler und Tester können an einem Modul arbeiten, ohne das Gesamtsystem zu beeinträchtigen. Das sorgt gerade in schnellen Entwicklungs- und Testzyklen für eine höhere Produktivität in der Entwicklung.

Allen Komponenten gemeinsam sind der grundsätzliche technische Aufbau und einige architekturelle Vorgaben, die insbesondere die Kommunikation über fachliche Grenzen hinweg festlegen (Datenaustausch und Zusammenarbeit von Benutzern). Als Beispiel seien die Kommunikation zwischen den Komponenten „*curriculum*" (verantwortlich für die Verwaltung und Darstellung von Studienplänen) und „*course*" (Verwaltung von Lehrveranstaltungen) genannt: Der Studienplan besteht aus in einer Baumstruktur gruppierten Einheiten, die jeder Student während seiner Studienzeit absolvieren muss. Für jede dieser Einheiten werden eine oder mehrere Lehrveranstaltungen geboten. Bei der Anzeige des Studienplans bedient sich *curriculum* über die Schnittstelle der in *course* gehaltenen Informationen über die Lehrveranstaltungen. So erhalten die Studierenden eine Gesamtsicht der angebotenen Vorlesungen. Umgekehrt verwendet *course* während des Ankündigungsprozesses für neue Lehrveranstaltungen Informationen aus *curriculum*, um einige Attribute der neuen Lehrveranstaltung mit passenden Daten zu füllen.

Diese Schnittstellen sind zum überwiegenden Teil als rein lesende Schnittstellen definiert. In genau dokumentierten Ausnahmefällen, vor allem bei fachlichen Überschneidungen, wird über diese Schnittstellen auch in „Fremdmodule" geschrieben.

Eine Sonderstellung nimmt die Komponente „*core"* ein: Sie stellt Funktionalitäten zur Verfügung, die von allen Komponenten benötigt werden, zum Beispiel Logging, Monitoring, Zugriff auf das Authentifizierungs- und Rollensystem und Templates für die Benutzeroberfläche. Durch die Verwendung gemeinsamer Templates, einer gemeinsamen Menüstruktur und durchgängig verwendeter Designvorgaben erscheint TISS dem Benutzer homogen.

Die Komponenten sind als *Java EE Applications* implementiert. Die in der folgenden Beschreibung genannten Technologien sind Schnittstellenspezifikationen, die aus der Spezifikation für *Java EE 6* stammen.

Die Architektur folgt einem Mehrschichtenmodell (4-Tier), und zwar die des Systems und jeder einzelnen Komponente (s. Abbildung 2). Die dort nicht gezeigte Schicht 0, die *Präsentationsschicht*, dient der Anzeige und Aufbereitung der Informationen für den User und zur Behandlung von Benutzereingaben. In TISS übernimmt der Webbrowser des Users diese Aufgaben; das dazu nötige HTML und JavaScript wird von *JavaServer Faces* generiert.

Abbildung 2 zeigt zwei Komponenten und ihre Interaktion und damit den Kern des Systems, genannt die *Logikschicht*. Hier ist die gesamte Anwendungslogik implementiert, aufgeteilt in die Teile Frontend (Aufbereitung und Verarbeitung der Daten für die Präsentationsschicht, Interaktion mit den Benutzern) und Backend (Geschäftsprozesse, Anwendungslogik). Die Trennlinie zwischen diesen beiden Teilen bildet eine *Serviceschicht* (API), über die die gesamte Kommunikation zwischen Front- und Backend läuft. Auch alle Mechanismen für die von dieser Komponente angebotenen Schnittstellen (API-EXT) sind hier vereint. Als grundlegendes Programmiermodel wird *CDI* verwendet (*Contexts and Dependency Injection for the Java EE Platform*, ebenfalls Teil der Java EE 6 Spezifikation).

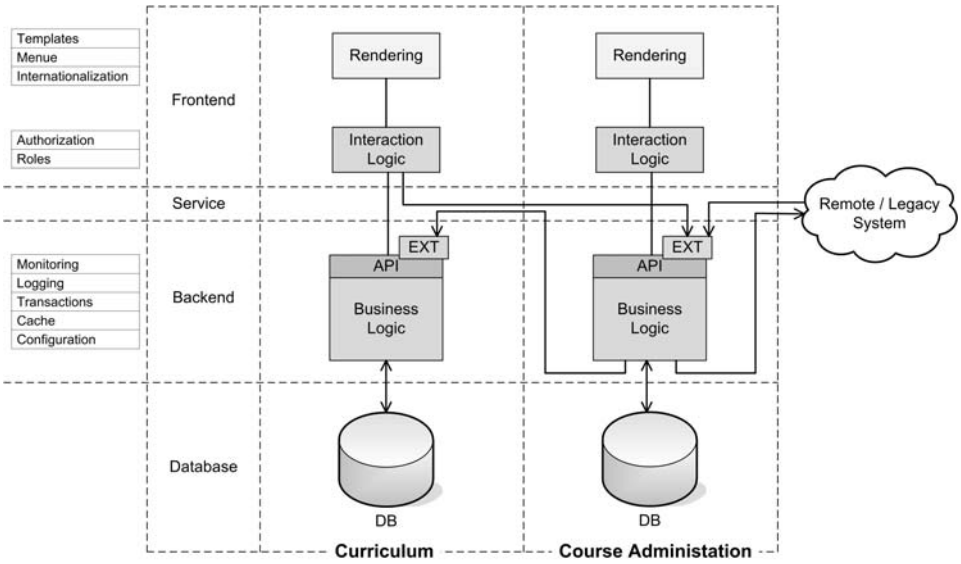


Abbildung 2: TISS Softwarearchitektur

Die *Datenhaltungsschicht* speichert die für den fachlichen Bereich einer Komponente relevanten Daten und übernimmt das Speichern und Laden von Daten für die Logikschicht dieser Komponente. Es gibt keine Zugriffsmöglichkeit der Datenhaltungsschichten zweier Komponenten untereinander; der gesamte Datenaustausch muss über die APIs stattfinden. Das sorgt für eine wirksame Entkopplung der einzelnen Datenbanken und für eine klare Festlegung der Datenhoheit für jede einzelne Tabelle. Auf Anwendungslogik auf DB-Ebene (zum Beispiel *Stored Procedures* oder DB *Trigger*) wird bewusst verzichtet. Die Abstraktion des Datenbankzugriffs erfolgt über *JPA* (*Java Persistence API*).

### 4.4 Betriebsarchitektur

Hohe Last, Datensicherheit, agile Weiterentwicklungsprozesse und Wartbarkeit bei gleichzeitigen Hochverfügbarkeitsanforderungen sind nur einige der Stresspunkte für den Betrieb von Softwaresystemen, denen durch Einsatz entsprechender Technologien

auch für den Betrieb entgegnet werden muss [Ka11]. Abbildung 3 zeigt das Grundkonzept der fehlertoleranten Infrastruktur, mit der TISS den genannten Punkten begegnet.

Jede Anfrage wird von einem Apache Webserver Cluster entgegen genommen. Statische Inhalte werden zur Entlastung des Backends aus einem Cache geladen, Anfragen auf dynamischen Inhalt werden an das Backend System weitergeleitet. Welcher der vorhandenen Application Server die Anfrage bearbeitet, wird von einem vorgeschalteten Load Balancer Cluster entschieden, der für eine gleichmäßige Verteilung der Last und damit für eine gute Auslastung der Server sorgt. Neben der Lastverteilung übernimmt der Load Balancer auch die zyklische Prüfung der Verfügbarkeit und leitet Anfragen bei Nicht-Erreichbarkeit auf andere Server um. Der Application Server wiederum prüft die Verfügbarkeit der Verbindung zum Datenbankserver. Kann dieser Application Server die Datenbank nicht erreichen, leitet der Load Balancer alle Anfragen auf einen anderen Application Server um.

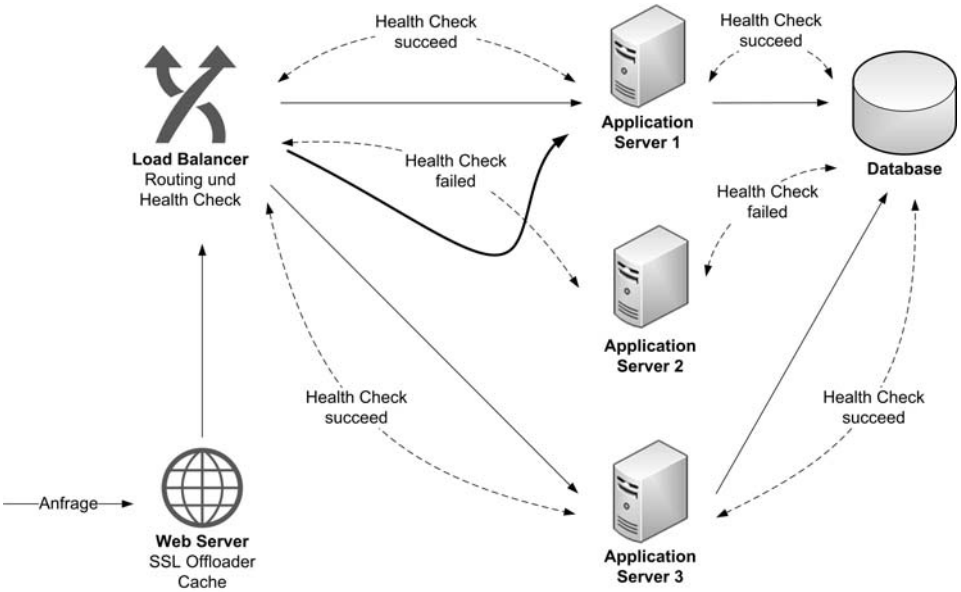


Abbildung 3: Hochverfügbarkeit durch Redundanz und Fehlertoleranz (nach [Ka11])

Im Falle eines Komplettausfalls aller Komponenten, beispielsweise auf Grund eines Feuers, ermöglicht eine Verteilung aller IT-Infrastrukturkomponenten auf zwei Standorte einen desaster-toleranten Betrieb. Im Ernstfall liegt die Wiederanlaufzeit im Bereich von einer Minute. Der mögliche Verlust von Daten bei einer derartigen Katastrophe wird durch den Einsatz einer zentralen Storage-Lösung verhindert, die die Daten zwischen beiden Standorten repliziert. Ein weiteres, wichtiges Merkmal der TISS-Betriebsarchitektur ist die Virtualisierung. So können Dienste de facto unabhängig von der darunter liegenden Hardware betrieben werden. Weitere Details zum Betrieb der TISS-Infrastruktur finden sich in [Ka11].

# 5 Kriterien für langfristige Lebensfähigkeit

Eingangs hatten wir vier Aspekte genannt, die wir für nachhaltige Software im Sinne von „langlebig bei gleichzeitiger Ressourcenschonung“ (s. Kap. 1) als essenziell betrachten. Dies wollen wir jetzt anhand der Aspekte Datenbasis, Software-Infrastruktur, Anforderungs- bzw. Changemanagement und Betriebskonzept näher beleuchten.

Bevor wir jedoch diese Punkte im Einzelnen behandeln, muss ein wesentlicher Bestandteil des Artefakts *Software* angesprochen werden, der meist fälschlicher Weise als „Dokumentation“ bezeichnet wird. Hierzu müssen wir kurz die Entstehungsgeschichte agiler Vorgehensmodelle betrachten. Es ändern sich ja *alle* Artefakte, nicht nur der Quellcode.

## 5.1 Die sogenannte Dokumentation

Ein CaMS gehört zur Klasse der *Embedded Systems*, die Lehmann *E-Programs* genannt hat [LP76, Le80]. Die Software ist nicht in ein technisches, sondern in ein soziotechnisches System eingebettet, gemeinhin *Organisation* genannt. Solche Softwaresysteme müssen sich evolutionär verändern können, weil ihre Umwelt dynamisch ist. Diesen Zusammenhang hat Christiane Floyd schon früh in dem unseres Wissens ersten evolutionären Entwicklungsmodell aufgezeigt [Fl81]. Dieses Modell hat viele Nachfolger in Form agiler Vorgehensmodelle gefunden. Eines davon war eine Arbeit, die sich früh mit der Frage der Bildung geeigneter Teilsysteme befasste und auch die sogenannte Dokumentation hinterfragte [Sp89, Kap. 3]. Das Buch wurde verfasst, als der Hype *Prototyping*[2] die Zeit der agilen Modelle einläutete. Das Thema machte es schon damals notwendig, die Rolle der sogenannten Dokumentation zu untersuchen.

Bei der Erstentwicklung ist jede denkbare Repräsentation (z. B. Use Cases, Prosa, Grafik, Testfälle) ein Teil der fachlichen und technischen Konstruktion des Systems und *keine* „Dokumentation“. Spätestens bei der Inbetriebnahme der Software müssen alle Bestandteile, besonders auch die der fachlichen Konstruktion daraufhin überprüft werden, ob sie mit der weiteren Evolution des Systems gepflegt werden können oder nicht. Auf dem Weg zum Quellcode gibt es viele redundante Bestandteile der fachlichen Konstruktion (s. [Sp89, S.78, 81ff]). Nur die essenziellen dürfen erhalten bleiben und müssen auf Dauer gepflegt werden. Geschieht das nicht, sind die Daten der sogenannten Dokumentation keine Information für die Entwickler, sondern *Desinformation*, die schädlicher sein kann als ein reiner, sprachlich und strukturell „sauberer“ Quellcode[3]. Hierauf wurde bei TISS geachtet: „*Alibi-Dokumentation ist noch schlechter als keine*“ [Su10]. Teile der Dokumentation wurden im Projektumfeld von TISS oft erst dann erstellt, wenn relevante Freeze-Zustände des Systems erreicht waren. Seriöse Dokumentation muss als wesentliche Voraussetzung für Nachhaltigkeit gesehen werden, allerdings nur, wenn sie die Gegenwart korrekt wiedergibt. Will man ältere Dokumente aufbewahren, nennt man sie *Archiv*.

[2] Als erste würden wir CASE ansehen, *Computer Aided Software Engineering* (s. auch [Gr10, S. 38]).
[3] Damit sind insbes. fachlich verständliche Variablen-, Prozedur- bzw. Methodennamen gemeint.

### 5.2 Die Datenbasis

Über die Schwierigkeiten, die eine schlecht entworfene Datenbasis mit sich bringt, wurde schon in Kap. 3 gesprochen. Daten sind diejenige Ressource des Produktionsfaktors Information – im Gegensatz zu Hardware und Software – die eine Organisation nicht kaufen kann. Die Leitlinien von TISS für eine nachhaltige Datenbasis waren:

(1) Jede Komponente hält ihre Daten in einer eigenen Datenbank. Dadurch entstehen fachlich zusammenhängende, aus wenigen Tabellen bestehende Datenbanken für die Einzelkomponenten.

(2) Datenbank-Schemata werden aus dem (gut dokumentierten) Programmcode generiert, und damit aus originären Quellen. Diese sind das Datenmodell, aber auch Quellcode-Templates. Auch Änderungen an Datenbank-Schemata werden so behandelt.

(3) Integritätsprüfungen werden in der Anwendungsschicht durchgeführt. So kann schon oberhalb der Datenbankebene ein hohes Maß an Datenintegrität erreicht werden. Dies macht von den sehr verschiedenen Implementierungen des Relationenmodells[4] unabhängig und lässt sich benutzerfreundlich implementieren.

(4) Die Basis für Variablennamen ist das logische Datenmodell, gerade auch in Quellprogrammen. Hierdurch erreicht man eine Homogenität der „Dokumente“ *Datenmodell*, *Datenbankschema* und *Quellcode* (s. hierzu auch: [Sp96]).

### 5.3 Die Software-Infrastruktur

Für die diversen Entscheidungen zur Software-Infrastruktur galten folgende Leitlinien:

(1) Implementierung nur gegen Spezifikationen und Standards.

(2) Hersteller-Unabhängigkeit.

(3) Offene Standards und Open-Source Basissoftware.

(4) Stabile Plattformen (Frameworks).

(5) Technische Prototypen bei neuen Technologien.

(6) Prinzip der „Austauschbarkeit“ im architekturellen Systemdesign.

Punkt (1) ist eigentlich eine softwaretechnische Binsenweisheit und bedarf hier keiner weiteren Erklärung.

[4] Als „relevante“ Systeme haben am Markt nur überlebt: DB2, Oracle, SQL-Server. Als „relevant“ bezeichnen wir die einzigen nach CC zertifizierten Systeme. Nur diese können professionellen Ansprüchen genügen. Vgl. https://www.bsi.bund.de/DE/Home/home_node.html

Dies ist anders bei Punkt (2), der leicht in einen dogmatischen Diskurs abdriftet. Es kann für kleine Organisationen sinnvoll sein, auf die Informationstechnik genau eines Herstellers zu setzen. Doch dies ist nicht unser Kontext. Ein CaMS für große Hochschulen ist sogar aus industrieller Sicht auf Grund der ungewöhnlich hohen Benutzerkomplexität ein äußerst anspruchsvolles System (siehe hierzu Abb. 2.5 in [Gr10]). Für große Systeme, die lange betrieben werden müssen, verbietet sich eine Herstellerbindung, insbesondere bei den Basis-Infrastrukturen Betriebssystem, Webserver und Datenbank. Wie an den Beispielen SAP und START in [BG10, S.72f] gezeigt wurde, darf noch nicht einmal eine Programmiersprache als langlebig genug angesehen werden. Da man aber eine Wahl treffen muss, ist eine standardisierte Sprache unabdingbar. Als Beispiel für die Hersteller-Unabhängigkeit sei die Datenbank gewählt. TISS läuft auf den Produktivsystemen mit einer Oracle DB, auf verschiedenen Testsystemen werden aber (ohne Änderungen am Code) MySQL beziehungsweise PostgreSQL verwendet. Die eingesetzte Datenbank ist also mehr eine Betriebs- und Lizenzfrage als ein die Infrastruktur prägendes Produkt. Dies schließt den punktuellen Einsatz herstellerspezifischer Module nicht aus, wenn sie offene Schnittstellen haben. So wird etwa ein Oracle-Modul im Produktivsystem benutzt, das Suchfunktionen für die Benutzer besonders effizient ausführt.

Punkt (3), offene Standards, hängt natürlich stark mit Punkt (2) zusammen. TISS setzt durchgängig auf Open-Source-Implementierungen von offenen Standards. Durch den Zugriff auf den Sourcecode dieser Projekte kann bei Bugs die Fehlerursache schnell gefunden und der entwickelnden Community gemeldet werden. Oft ist es möglich, den Fehler selbst zu beheben. Bei wichtigen Bugfixes ist es sogar denkbar, eine selbst gefixte Variante des fehlerhaften Codes zu verwenden, bis der Hersteller eine fehlerbereinigte Version bereitstellt. Nicht zuletzt eröffnet eine aktive Beteiligung von Teammitgliedern in der Open-Source-Community die Möglichkeit, die Entwicklung der Software mitzugestalten und so für das eigene Projekt wichtige Features schnell zu integrieren und Mitspracherechte in der Weiterentwicklung zu erhalten.

Die Verwendung von Frameworks, unser vierter Punkt, ist schon lange Stand der Technik (s. [Sp89, Kap. 10], [Gr10, Kap. 6]). Ohne ein Framework (oder mehrere, komponenten- und ebenenspezifische) lässt sich kein System über viele Jahre stabil halten. Ohne Framework droht schnell *Architectural Decay* (Erosion der Struktur in der Evolution). Es versteht sich, dass für TISS gerade im Bereich der Architektur auch technische Prototypen (s. [Sp89, Kap.7]) evaluiert und die am besten geeigneten ausgewählt wurden. Das frühe Herstellen von echten betrieblichen „Durchstichen“ in das Zielsystem verbessert die prinzipielle Betriebskonformität nachhaltig. Dies ist unser Punkt (5).

Punkt (6): Entwicklungs- und Betriebsarchitektur wurden bei TISS nachhaltig so gestaltet, dass die Kernkomponenten alle prinzipiell austauschbar sind, ohne dass das Gesamtsystem auszutauschen ist. Eine mittelfristige Bindung an einen Hersteller, an ein Produkt, an eine Sprache bedeutet somit nicht die zwangsweise Herstellung eines langfristigen Software-Monolithen. Ein solcher „kaum entwirrbarer“ Monolith war mit den Altsystemen an der TU Wien gegeben und sollte durch dieses Architekturprinzip verhindert werden.

### 5.4 Die Einbindung der Organisation – Anforderungsmanagement

Gerade bei einem System mit hoher Benutzerkomplexität und Nutzerklassen mit sehr unterschiedlichen Anforderungen (etwa Dozenten, interne Verwaltung und Studierende) ist die Einbindung der Akteure der Organisation essenziell. „Eindeutige Geschäftsprozesse“ lassen sich leicht einfordern [FH09], sind auch notwendig, aber schwer durchsetzbar. Historisch etablierte Abläufe müssen hinterfragt und gegebenenfalls neu geordnet und durch elektronische Workflows unterstützt oder abgelöst werden. Dieser einschneidende Veränderungsprozess muss von allen Akteuren mitgetragen werden; nur mit deren Mitwirkung ist die Neugestaltung möglich. *Befehlen* ist nicht nur unmöglich, sondern kontraproduktiv[5].

Der folgende Unterschied eines CaMS zu einem industriellen Anwendungssystem wiegt schwer: Beim Unternehmens- oder Behördensystem gehört der Kunde zur Umwelt, beim CaMS ist er Systemelement. Ein Student ist in einer Hochschule *kein* klassischer „Kunde“. Er ist jedoch ein Nutzer, der sehr ernst genommen werden muss und z. B. in Österreich ein Nutzer, der oft deutlich höhere Anforderungen einbringen kann, als es klassische Angestellte oder typische Kunden tun würden. Als Beispiel seien die Datenschutz- und Privacy-Erfordernisse an den Universitäten und z. B. die e-Voting-Debatte in Österreich 2010 genannt.

Bei Beschäftigten, die Veränderungen weniger gewohnt sind als Mitarbeiter in der Wirtschaft, erlebt man schnell den Effekt, dass die Software sich zunehmend zum Sündenbock für negativ bewertete Veränderungen entwickelt, die andere organisatorische Ursachen haben [Ja09]. Zudem wird vielen Anwendern erst in der praktischen Anwendung bewusst, welche Implikationen ihre fachlichen Vorstellungen auf die technischen Prozesse, Abläufe und die Administration haben [De09]. An der TU Wien ging der Einführung von TISS eine sehr intensive Phase der Anforderungsanalyse voraus. Bewährte Kernfunktionalitäten des Altsystems sollten auf einer dem State of the Art entsprechenden technologischen Basis nachgebildet und erweitert werden. Hierzu erfolgte zum Teil eine (Re-)Dokumentation der alten Verfahren mit zahlreichen Workshops und Interviews. *Akzeptanz* verlangt, dass man erst einmal eine gemeinsame Sprache findet.

Bei Hochschulen mit einer traditionell großen Autonomie der Substrukturen darf man nicht unterschätzen, dass die Entscheidungs- und damit die Entwicklungsprozesse zur Umsetzung spezifischer Funktionalitäten iterativ und oftmals auch langwierig sind. Während der Entwicklung eines Systems müssen mehrere 10.000 Einzelentscheidungen getroffen werden, über die eine Führung nicht bis ins letzte Detail informiert sein kann. Erfahrene Analysten und eine starke Projektleitung, die von der Hochschulleitung unterstützt wird, sind für diese zeitkritischen Entscheidungsprozesse unerlässlich. Diese Prozesse waren für das Projekt TISS und sind auch jetzt für einen reibungslosen Betrieb ein wesentlicher Erfolgsfaktor.

TISS hatte trotz guter Vorplanung starke, für die User teilweise anstrengende „Einführungsschmerzen“ zu vermerken, weniger auf Seite der studentischen Nutzergemeinschaft

[5] Zitat einer Psychologin, das die soziale Dimension des Problems auf den Punkt bringt:
*„Entscheidungen, die nicht getragen werden, werden unterlaufen.“*

als auf Seite der Lehrenden und Verwaltungsmitarbeiterinnen. Letzteres war nicht auf mangelnde Methodik sondern auf die strategische Entscheidung des Rektorats zu einer um ein Jahr vorgezogenen Abschaltung des Altsystems mit der damit verbundenen Einführung eines Roh-Produktes TISS zurückzuführen, das erst im Zuge der ersten 4 Monate des Betriebes fertiggestellt wurde. Viel Stress für die Kern-User aber auch eine beträchtliche Ersparnis für die Universität durch den Wegfall von Wartungsarbeiten für ein Jahr am Altsystem sowie eine deutliche Verdichtung der Entwicklungsarbeiten am Neusystem.

### 5.5 Der Betrieb des Systems

Mit Verbreiterung der Funktionalitäten und steigender Anzahl der Nutzer muss nicht nur die softwaretechnologische Basis sondern auch der Betrieb eine entsprechende Skalierbarkeit erlauben. Besonders hohe Erwartungen an die Verfügbarkeit eines Systems werden naturgemäß dann gestellt, wenn die Anzahl der gleichzeitigen Nutzer am höchsten ist und diese Nutzer für sie kritische Aktionen durchführen müssen. Ein angemessenes Betriebskonzept muss auf die erwarteten Lastspitzen ausgelegt werden, um einen reibungslosen Betrieb mit entsprechender Verfügbarkeit des Systems zu ermöglichen. Bei einem CaMS werden diese Lastspitzen zu Beginn (und ggf. am Ende) eines Semesters erreicht. Abbildung 4 zeigt die Anzahl der Seitenaufrufe pro Stunde im Bereich Lehre zu Beginn des Sommersemesters 2011 – bei Lastspitzen wurden bis zu 12.000 Seitenaufrufen pro Minute verarbeitet.

Der Kampf um beschränkte Plätze bei Lehrveranstaltungen beginnt, Anmeldungen zu mehreren tausend Lehrveranstaltungen finden innerhalb kurzer Zeit statt, oft zeitgleich. Ein wesentlicher Vorteil der in Abschnitt 4.4. dargestellten Betriebsarchitektur ist ihre gute horizontale Skalierbarkeit. Erhöhen sich die Anforderungen an die Performance zu Semesterbeginn, können bei Bedarf einfach und ohne Betriebsunterbrechung zusätzliche Application Server kurzfristig integriert und so die steigende Anzahl von Anfragen auf mehrere Server verteilt werden.

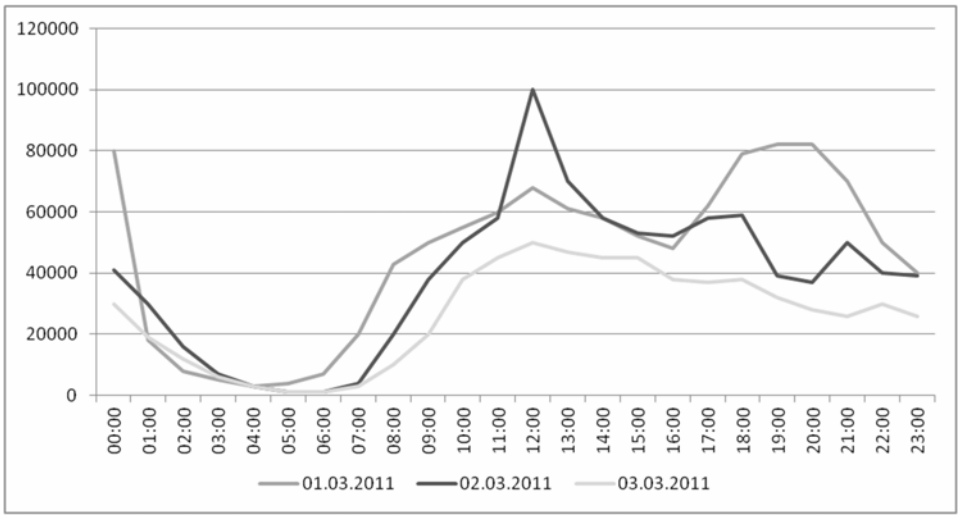


Abbildung 4: Seitenaufrufe pro Stunde im Bereich Lehre zu Semesterbeginn (aus [Ka11])

Vor der Inbetriebnahme von TISS war der Semesterbeginn an der TU Wien von stunden- und teilweise tagelangen Systemausfällen auf Grund von Überlastungen geprägt. Weder die Software- noch die Hardwarearchitektur waren für derartige Lastspitzen ausgelegt, alle damals verfügbaren Mittel zur Erhöhung der Verfügbarkeit waren ausgeschöpft. Mit der Inbetriebnahme von TISS erlebten die Studierenden und Mitarbeiter der TU Wien das erste Mal eine unterbrechungsfreie Serviceverfügbarkeit zu Semesterbeginn.

### 5.6 Ergebnis zur Nachhaltigkeit

Wir hatten in diesem Kapitel die Eigenschaften und Entwicklungshistorie von TISS untersucht, die Nachhaltigkeit im Sinne unserer Arbeitsdefinition unterstützen. Dabei hatten wir festgestellt, dass ein in eine Organisation eingebettetes Softwaresystem *nachhaltig* genannt werden kann, wenn es:

- die Organisationsziele dauerhaft unterstützt und nicht behindert,
- über lange Zeit (> 20 Jahre) evolutions- und informationsfähig[6] bleibt,
- möglichst unabhängig von den Entscheidungen einzelner liefernder Akteure ist,
- technologisch allgemein akzeptiertem Fortschritt folgen kann.

[6] Längsschnitt-Daten müssen auch nach 10 Jahren noch korrekt sein und nicht – wie den Autoren bekannte, gängige Systeme in Hochschulen – plötzlich Nullwerte in Tabellenspalten enthalten. Dann liefern Datenbankanfragen bekanntlich falsche Ergebnisse. Dieses Problem löst auch kein Werkzeug, etwa ein Data Warehouse. Ein Beispiel aus der TISS-Entwicklung wurde in Kap. 2 berichtet.

Zwischen Hardware und Software bestehen hinsichtlich Nachhaltigkeit zwei fundamentale Unterschiede:

(1) Hardware ist sehr viel kurzlebiger. Durch den raschen technologischen Fortschritt sind die sinnvollen Nutzungszeiten erheblich kürzer als bei Software.

(2) Als materielles Gut spielt bei Hardware die Frage der Wiederverwendung materieller Ressourcen eine wichtige Rolle, bei Software fast keine.

Wir fassen zusammen:

> Unter **Nachhaltigkeit** eines technischen Systems verstehen wir seine Nützlichkeit für die Akteure über lange Zeiträume zu günstigen Kosten.

## 6 Ausblick und Weiterentwicklung

Die gewonnen Erfahrungen, die die TU Wien bei der Einführung und der seither laufend stattfindenden Weiterentwicklung von TISS machen durfte, bestätigen, dass die unter Kapitel 5 genannten Faktoren für den Erfolg eines CaMS essentiell sind. Schon in [Kl09] wird darauf hingewiesen, dass *„organisatorische, personelle und technologische Faktoren für ein erfolgreiches Informationsmanagement gleichermaßen berücksichtigt werden müssen“* und *„Die Akzeptanz der Informationssysteme steigt, wenn sie in die Organisation eingebettet werden“*. Die TU Wien wird den mit der Einführung von TISS gewählten und seither ausgezeichnet bewährten Kurs stetig weiter verfolgen. Das System wird weiter wachsen – ggf. auch schrumpfen –, kurzum, es wird laufend an die Bedürfnisse der Hochschule und der Nutzer angepasst werden. Dank der stabilen und flexiblen technologischen Basis, der offenen, standardisierten Schnittstellen und der etablierten Projektstrukturen ist TISS als Kernsystem der TU Wien auf zukünftige Anforderungen vorbereitet. Verstärkte Unterstützung mobiler Services, z. B. die Integration von *Near Field Communication* (NFC)-Technologien oder die Anbindung weiterer (externer) Systeme werden dabei ebenso eine Rolle spielen wie die laufende Verbesserung durch Einarbeitung des Benutzerfeedbacks.

## Literaturverzeichnis


[AGW05] Alpar, P.; Grob, H.L.; Weimann, P.; Winter, R.: Anwendungsorientierte Wirtschaftsinformatik. 4. überarb. und erw. Aufl., Vieweg, Braunschweig – Wiesbaden, 2005.

[Bo09] Bode, A. et.al.: Integrierte Campus-Managementsysteme. In: [HKF09], S. 451-552.

[BD04] Brügge, B.; Dutoit, A.H.: Objektorientierte Softwaretechnik. Pearson Studium, München, 2004.

[BGS10] Bick, M.; Grechenig, T.; Spitta, T.: Campus-Management-Systeme – Vom Projekt zum Produkt. In: *Pietsch, W.; Krams, B.* (Hrsg.): Vom Projekt zum Produkt. Fachtagung Aachen, Dez. 2010, Lecture Notes in Informatics 178, Koellen, Bonn, S. 61-78.

[Br09] Brune, H. et.al.: Ein Campus-Management-System als evolutionäre Entwicklung. In: [HKF09], S. 483-492.

[De09] Degenhardt, L. et.al.: Campus-Management-Systeme erfolgreich einführen. In: [HKF09], S. 463-472.
[FH09] Fischer, H.; Hartau, C.: STiNE an der Universität Hamburg zur Einführung eines integrierten Campus Management Systems. In: [HKF09], S. 533-542.
[Fl81] Floyd, C.: A Process-oriented Approach to Software Development, in: Systems Architecture, Proc. of the 6th European ACM Regional Conference. Westbury House 1981, pp.285-294.
[Gr10] Grechenig, T. et.al.: Softwaretechnik – Mit Fallbeispielen aus realen Entwicklungsprojekten. Pearson Studium, München, 2010.
[HKF09] Hansen, H.R.; Karagiannis, D.; Fill, H-G. (Hrsg.): Business Services – Konzepte, Technologien, Anwendungen (Bd 2). 9. Int. Tagung Wirtschaftsinformatik, Febr. 2009 Wien.
[Ja09] Janneck, M. et.al.: Von Eisbergen und Supertankern: Topologie eines Campus-Management-Einführungsprozesses. In: [HKF09], S. 453-462.
[Kl08] W. Kleinert, T. et.al.: The Making of TISS. ZIDline, Nr. 18, Juli 2008, S. 3-8.
[Kl09] Klug, H.: Erfolgsfaktoren bei der Umstellung von Informationssystemen an Hochschulen. In: [HKF09], S. 473-482.
[Ka11] Kamenik, R. et.al.: Betrieb der TISS-Infrastruktur. ZIDline, Nr. 23, April 2011, S. 14-18.
[Le80] Lehmann, M.M.: Programs, Life Cycles and Laws of Software Evolution, in: IEEE Proceedings 68(1980) 9, pp.1060-1076.
[LP76] Lehman, M.M., Parr, F.N.: Program Evolution and its Impact on Software Engineering, in: 2nd ICSE, San Francisco 1976, 350-357.
[MWF08] Murer, S.; Worms, C.; Furrer, F.J.: Managed Evolution – Nachhaltige Weiterentwicklung großer Systeme. Informatik Spektrum 31(2008) 6, S. 537-547.
[Pf01] Pfleeger, S.L.: Software Engineering. 2nd ed. Prentice Hall, Upper Saddle River 2001.
[SB08] Spitta, T.; Bick, M.: Informationswirtschaft. 2. überarb. und erw. Aufl., Springer, Berlin – Heidelberg, 2008.
[So01] Sommerville, I.: Software Engineering. 6th ed., Addison-Wesley, Boston et al. 2001.
[Sp89] Spitta, T.: Software Engineering und Prototyping. Springer, Berlin et al. 1989.
[Sp96] Spitta, T.: „CASE“ findet im Kopf statt. INFORMATIK/INFORMATIQUE 3(1996) 3, 17-25.
[St09] Strobl, S. et.al.: Digging Deep: Software Reengineering supported by Database Reverse Engineering of a System with 30+ Years of Legacy. Wien, Vienna University of Technology (TU Wien), 2009.
[Su10] Suppersberger, M. et.al.: TISS Epistemologie II. ZIDline, Nr. 22, Juni 2010, S. 16-21.
[Ve90] Vetter, M.: Aufbau betrieblicher Informationssysteme mittels konzeptioneller Datenmodellierung, 6. Aufl., Teubner, Stuttgart, 1990.